Formulating and Solving Nonlinear Programs as Mixed Complementarity Problems*

Michael C. Ferris[1] and Krung Sinapiromsaran[1]

University of Wisconsin, Computer Sciences Department,
1210 West Dayton Street, Madison, Wisconsin 53706

Abstract. We consider a primal-dual approach to solve nonlinear programming problems within the AMPL modeling language, via a mixed complementarity formulation. The modeling language supplies the first order and second order derivative information of the Lagrangian function of the nonlinear problem using automatic differentiation. The PATH solver finds the solution of the first order conditions which are generated automatically from this derivative information. In addition, the link incorporates the objective function into a new merit function for the PATH solver to improve the capability of the complementarity algorithm for finding optimal solutions of the nonlinear program. We test the new solver on various test suites from the literature and compare with other available nonlinear programming solvers.

Keywords: Complementarity problems, nonlinear programs, automatic differentiation, modeling languages.

1 Introduction

While the use of the simplex algorithm for linear programs in the 1940's heralded the inception of operations research as a practical discipline, the extension of the field to nonlinear programs (NLP) has been much more recent. The theory of NLP was extensively developed in the 1950's and 60's, culminating perhaps with the landmark books [12,25]. Leaving aside unconstrained optimization, practical algorithms for constrained nonlinear optimization rivaling the simplex algorithm were much slower to develop. In fact, the MINOS code [26] released in 1976 was the first code that could deal reliably with problems of relatively large size.

The advent of modeling languages [3,14] allowed these solvers to be used by modelers that were not operation research or numerical analysis specialists. Modeling languages allow optimization problems to be communicated to solvers in an efficient form, carrying out data manipulations, generation of multiple sets of indexed equations, exploiting simple constraint types and

* This research was partially supported by National Science Foundation Grant CCR-9619765 and Air Force Office of Scientific Research Grant F49620-98-1-0417.

converting problems to the format required by a solver without modeler intervention. Furthermore, computational advances such as the use of automatic differentiation techniques [19,18,28] to generate the first order derivatives of the nonlinear functions can be used directly in a solver implementation. Currently, GAMS [3] and AMPL [14] are used in a large variety of applications. Most of the commercially available solvers for linear and nonlinear programs can be used directly from one or both of these systems.

The 1980's and 1990's have generated two significant algorithmic changes to the field. The first major change was the introduction of interior point methods for linear programming by Karmarkar [21] in 1984, as a practical alternative to the theoretically important polynomial time ellipsoid algorithm of Khachian [23]. The idea has been considerably developed; currently it appears that primal-dual methods are the most effective in large scale linear programming settings [34].

In nonlinear programming, a significant improvement has been observed for non-convex problems by using second order information. While Quasi-Newton methods can be used for problems whose feasible region lies in a relatively small dimension subspace, and limited memory methods are effective for unconstrained and bound constrained problems, it is becoming increasingly clear that methods that exploit second order information (either using negative curvature within a trust region or line search framework) are more efficient and robust. Unfortunately, it is only recently [15] that second order information has become available from a modeling language, namely AMPL.

This paper is an attempt to combine some of the features of these last two improvements. The idea is to use a primal-dual framework for NLP in conjunction with second order information. We first start with the first order conditions of the original NLP model in Section 2.1, which we cast as a mixed complementarity problem (MCP) in Section 2.2. In Section 3, we explain the PATH solver implementation for MCP and its requirements and describe the use of a merit function to solve the MCP problem. Then we introduce a new merit function associated with solving NLP's. Section 4 gives details of our NLP solver, PATHNLP, with the MCP function evaluation and its Jacobian being evaluated by AMPL. In particular, we show how second order information of the NLP is utilized via solver link libraries in Section 4.2.

Section 5 gives numerical results for our approach on a set of nonlinear test problems extracted from the AMPL web site. Specifically, we test all models in the Hock/Schittkowski test suite [20] and compare the results of the PATH solver with LANCELOT [4], MINOS [27], NPSOL [17] and SNOPT [16]. Other large scale examples, including problems from portfolio and structural optimization are also tested. We believe these results indicate this is already a promising approach and warrants further investigation in the future.

2 Mathematical formulation

In this paper, we concentrate on the following constrained nonlinear program

$$\begin{array}{l} \text{minimize} \quad f(x) \\ \text{subject to } g(x) \leq 0,\ h(x) = 0,\ x \in B, \end{array} \tag{1}$$

where $f : \mathbf{R}^n \mapsto \mathbf{R}, g : \mathbf{R}^n \mapsto \mathbf{R}^m$ and $h : \mathbf{R}^n \mapsto \mathbf{R}^p$ are twice continuously differentiable, and $B := \{x \in \mathbf{R}^n | r \leq x \leq s\}$ with $r_i \in [-\infty, \infty]$ and $s_i \in [r_i, \infty]$. Let $S := \{x \in B | g(x) \leq 0, h(x) = 0\}$ denote the feasible region. We will focus on finding a point that satisfies the first order conditions of the NLP (1).

2.1 The first order conditions of NLP

The concept of the Lagrangian function and the Lagrange multipliers play a crucial role in defining a first order point for the NLP (1). The Lagrangian function is a weighted summation of the objective function and the constraint functions, defined as follows

$$L(x, \lambda, \nu) := f(x) - \lambda^T g(x) - \nu^T h(x),$$

where λ and ν denote the Lagrange multipliers (dual variables) corresponding to the inequality and equality constraints, respectively.

The first order necessary conditions for the NLP (1) are

$$\begin{array}{c} 0 \in \nabla_x L(x, \lambda, \nu) + N_B(x) \\ 0 \geq \lambda - g(x) \leq 0 \\ h(x) = 0, \end{array} \tag{2}$$

where $N_B(x) = \{z \in \mathbf{R}^n | (y - x)^T z \leq 0, \forall y \in B\}$ is the normal cone [32] to B at x.

In the case that r_i or s_i is finite, the definition of the normal cone allows the first equation of (2), to be rewritten in the following manner. If $x_i = r_i$, then

$$\left(\nabla_x L(x, \lambda, \nu)\right)_i \geq 0,$$

while if $x_i = s_i$, then

$$\left(\nabla_x L(x, \lambda, \nu)\right)_i \leq 0$$

and for any values of r_i and s_i, if $r_i < x_i < s_i$, then

$$\left(\nabla_x L(x, \lambda, \nu)\right)_i = 0.$$

These conditions coupled with the regularity condition on the point x establish the necessary conditions for NLP which are normally called the Karush-Kuhn-Tucker (KKT) conditions [22,24]. Whenever the Hessian matrix of the Lagrangian function is positive definite at (x^*, λ^*, ν^*), the first order conditions are also sufficient for x^* to be a strict local minimizer of NLP.

2.2 Primal-dual formulation of NLP

The standard mixed complementarity problem (MCP) is defined as the problem of finding a point $z \in \mathbf{R}^n$ inside the box $B = \{z | -\infty \leq l < z < u \leq \infty\}$ that is complementary to a nonlinear function $F : \mathbf{R}^n \to \mathbf{R}^n$. We assume without loss of generality that $l_i < u_i$ for all $i = 1, 2, \ldots, n$.

The point z is complementarity to $F(z)$ when

$$\begin{array}{lll} \text{either } z_i = l_i & \text{and } F_i(z) \geq 0 & \\ \quad \text{or } z_i = u_i & \text{and } F_i(z) \leq 0 & \text{for } i = 1, \ldots, n \\ \quad \text{or } l_i < z_i < u_i & \text{and } F_i(z) = 0. & \end{array}$$

If $l \equiv -\infty$ and $u \equiv \infty$, MCP becomes the problem of finding a zero of a system of nonlinear equations, that is to find $z \in \mathbf{R}^n$ such that $F(z) = 0$, while if $l = 0$ and $u \equiv \infty$, the problem is the Nonlinear Complementarity Problem (NCP) of finding $z \in \mathbf{R}^n$ such that $z_i \geq 0, F_i(z) \geq 0$ and $z_i F_i(z) = 0$, for all $i = 1, \ldots, n$. The latter property $z_i F_i(z) = 0$ is often called complementarity between z_i and $F_i(z)$.

Let z be composed of the primal variable x and the dual variables λ and ν of the NLP (1). The nonlinear MCP function can be written as a vector function of the first order derivative evaluation of the Lagrangian function with respect to the corresponding primal and dual variables that is

$$F(z) := \begin{bmatrix} \nabla_x L(z) \\ -\nabla_\lambda L(z) \\ -\nabla_\nu L(z) \end{bmatrix}.$$

The nonlinear MCP model is to find $z = (x, \lambda, \nu) \in \mathbf{R}^q$ where $q = n+m+p$ that is complementary to the nonlinear vector function F from $\mathbf{R}^q \mapsto \mathbf{R}^q$ given above along with lower bounds l and upper bounds u

$$F(z) = \begin{bmatrix} \nabla_x L(z) \\ g(x) \\ h(x) \end{bmatrix}, l := \begin{bmatrix} r \\ -\infty \\ -\infty \end{bmatrix}, u := \begin{bmatrix} s \\ 0 \\ \infty \end{bmatrix}. \tag{3}$$

Here $\begin{aligned} \nabla_x L(z) &= \nabla_x f(x) - \lambda^T \nabla_x g(x) - \nu^T \nabla_x h(x) \\ &= \nabla_x f(x) - \sum_{i=1}^m \lambda_i \nabla_x g_i(x) - \sum_{j=1}^p \nu_j \nabla_x h_j(x). \end{aligned}$

By comparing the MCP (3) to the KKT conditions (2), it is clear that this formulation is equivalent to the first order conditions of the NLP (1). This simple observation allows us to solve the NLP problem using an MCP solver, which is the subject of Section 4.

3 The PATH solver and merit functions

The PATH solver [6] is a nonsmooth Newton type algorithm [31] which finds a zero of the normal map [30]

$$F_+(x) := F(\pi(x)) + x - \pi(x),$$

where $\pi(x)$ is the closest point in B to the variable x in the Euclidean norm. It is well known [30] that finding a zero of this normal map is equivalent to solving MCP. In particular if x is a zero of the normal map, then $\pi(x)$ solves MCP, while if z solves MCP then $z - F(z)$ is a zero of the normal map.

3.1 Overview of the algorithm

The essential idea of the code is to linearize the normal map $F_+(x)$ about the current iterate to obtain a piecewise linear map whose zero is sought using a homotopy approach [7]. To monitor progress in the nonlinear model, a nonmonotone path-search is used [29]. Recent extensions [9] have introduced a function Ψ to be used in conjunction with the code, both as a residual and a merit function.

The following pseudo code shows the main algorithm steps of the PATH solver to find a KKT point

Loop until $\Psi(x)$ is less than a convergence tolerance {

Solve the linearization of the MCP problem to obtain the Newton point;

Search the path between the current point and the Newton point.

If the new point gives rise to a better value for the merit function then accept it.

Otherwise use the merit function to find a descent direction and search along this direction for a new point.

}

Details on the solution of linearization and the path-search mechanism can be found in [6,10]. In this paper, we just indicate the changes specific to solving NLP's. The Newton-type PATH solver uses the Jacobian matrix of the MCP function (3) to find its path-searching direction. In the above context, the Jacobian matrix is computed by finding the derivative of the MCP function. It uses the first and second order derivatives of the original NLP objective function and constraints as

$$\nabla_z F(z) := \begin{bmatrix} \nabla^2_{xx} L(x,\lambda,\nu) & -\nabla^T_x g(x) & -\nabla^T_x h(x) \\ \nabla_x g(x) & 0 & 0 \\ \nabla_x h(x) & 0 & 0 \end{bmatrix},$$

where $\nabla^2_{xx} L(x,\lambda,\nu) = \nabla^2_{xx} f(x) - \sum_{i=1}^{m} \lambda_i \nabla^2_{xx} g_i(x) - \sum_{j=1}^{p} \nu_j \nabla^2_{xx} h_j(x)$.

3.2 The merit function for the PATH solver

The most recent version of the PATH solver [9] does not use the residual of the normal map for a merit function. Instead, it utilizes the Fischer-Burmeister function [13] defined as the mapping $\phi : \mathbf{R}^2 \to \mathbf{R}$,

$$\phi(p, q) := \sqrt{p^2 + q^2} - p - q,$$

where p and q are scalar variables. This function exhibits the complementarity property when the function value is zero, that is

$$\phi(p, q) = 0 \text{ if and only if } p \geq 0, q \geq 0 \text{ and } pq = 0.$$

For the MCP problem, the residual and merit function used is $\Psi : \mathbf{R}^n \to \mathbf{R}$,

$$\Psi(x) := \frac{1}{2}\psi(x)^T\psi(x),$$

where $\psi(x)$ is the Fischer operator [1] defined in (4) from $\mathbf{R}^n$ to $\mathbf{R}^n$ that maps x_i and $F_i(x)$ as parameters to the Fischer-Burmeister function componentwise as follows:

$$\psi_i(x) := \begin{cases} \phi(x_i - l_i, F_i(x)) & \text{if } -\infty < l_i \leq x_i < \infty, \\ -\phi(u_i - x_i, -F_i(x)) & \text{if } -\infty < x_i \leq u_i < \infty, \\ \phi(x_i - l_i, \phi(u_i - x_i, -F_i(x))) & \text{if } -\infty < l_i \leq x_i \leq u_i < \infty, \\ -F_i(x) & \text{if } -\infty < x_i < \infty. \end{cases} \tag{4}$$

This function is nonnegative and is zero at the solution point. A key feature for its use as a merit function is its continuously differentiability. It allows gradient steps to be used when the path-searching direction does not lead to a descent direction.

The nonlinear MCP function (3) from Section 2.2 contains only the first order derivatives of the objective function and constraints. The formulation exhibits the deficiency of finding KKT points for NLP. In an effort to avoid this deficiency, we introduce a new merit function for the PATH solver that explicitly incorporates the objective function. We now describe the implementation of the new merit function and give some computational results in Section 5.

The PATH solver uses a merit function to find a gradient descent direction when its Newton direction fails to find a descent direction. It uses the residual function $\Psi(x)$ to identify the stopping criteria. We define a new merit function for the PATH solver applied to NLP's which is a weighted average of the residual function Ψ and the objective function f as

$$\varphi(x) = (1 - \gamma)\Psi(x) + \gamma f(x),$$

where $\gamma \in [0, 1]$.

When γ is equal to zero, $\varphi(x) = \Psi(x)$ entreating the original PATH solver to satisfy the first order conditions of the NLP problem. For $\gamma > 0$, the objective function affects the search direction. However, if the weighted value of the objective function reaches 1, then a solution is not guaranteed to satisfy the first order conditions. With appropriate choice of γ, our new merit function guides the path-searching algorithm to escape KKT points that are not local minimizers of the original NLP. After our experimentation with the value of γ, we decided to take a fixed value of $\gamma = 0.3$ for the purposes of the results given in Section 5.

In the next section, we show how the NLP model in AMPL is automatically modified and transformed into the MCP formulation. The MCP function (3) and its Jacobian evaluation are specified in more detail.

4 The PATHNLP solver for AMPL nonlinear programs

To solve the NLP problem in AMPL, a user could specify the complementarity formulation directly using the AMPL language [8]. This would require a modeler to write down explicitly the first order conditions as detailed in Section 2.2. This process is very cumbersome and prone to error. In this paper, we propose to use the AMPL solver library to take an NLP specified directly in AMPL and form the required F and its Jacobian matrix for the PATH solver automatically within the solver link. This means that a modeler simply has to change the solver name in order to use the approach outlined in this paper.

4.1 MCP formulation from AMPL

The NLP problem passed to a solver from the AMPL environment is defined as

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to } a \le c(x) \le b,\ r \le x \le s, \end{aligned}$$

where $f : \mathbf{R}^n \mapsto \mathbf{R}, c : \mathbf{R}^n \mapsto \mathbf{R}^m$ with $a, b \in \mathbf{R}^m$ and $x, r, s \in \mathbf{R}^n$.

We now show how to recover the NLP format (1) as described in Section 2 from the data given above. We define five mutually exclusive index subsets of an index set $I = \{1, 2, \ldots, m\}$ of the constraint function c as

$$\begin{aligned} \mathcal{L} &:= \{i \in I | -\infty < a_i \text{ and } b_i \equiv \infty\} \\ \mathcal{U} &:= \{i \in I | a_i \equiv -\infty \text{ and } b_i < \infty\} \\ \mathcal{E} &:= \{i \in I | -\infty < a_i = b_i < \infty\} \\ \mathcal{R} &:= \{i \in I | -\infty < a_i < b_i < \infty\} \\ \mathcal{F} &:= \{i \in I | a_i \equiv -\infty \text{ and } b_i \equiv \infty\}, \end{aligned}$$

where $\mathcal{L}$ is the index set of lower bound constraints, $\mathcal{U}$ is the index set of upper bound constraints, $\mathcal{E}$ is the index set of equality constraints, $\mathcal{R}$ is the index set of range constraints, and $\mathcal{F}$ is the index set of free constraints.

The NLP model from AMPL is therefore rewritten as

$$\begin{array}{lrll} \text{minimize} & f(x) & & \\ \text{subject to } a_i \leq & c_i(x) & & i \in \mathcal{L} \\ & c_i(x) \leq & b_i & i \in \mathcal{U} \\ & c_i(x) = & a_i & i \in \mathcal{E} \\ a_i \leq & c_i(x) \leq & b_i & i \in \mathcal{R} \\ & c_i(x) & \text{is free} & i \in \mathcal{F} \\ r \leq & x & \leq s. & \end{array}$$

Define $y \in \mathbf{R}^{|\mathcal{R}|}$ as artificial variables for each range constraint, where $|\mathcal{R}|$ is the number of range constraints. Then by dropping the free constraints, the model is equivalent to

$$\begin{array}{lrll} \text{minimize} & f(x) & & \\ \text{subject to} & a_i - c_i(x) & \leq 0 & i \in \mathcal{L} \\ & c_i(x) - b_i & \leq 0 & i \in \mathcal{U} \\ & c_i(x) - a_i & = 0 & i \in \mathcal{E} \\ & c_i(x) - y_{j_i} & = 0 & i \in \mathcal{R} \\ & a_i \leq y_{j_i} \leq b_i & & i \in \mathcal{R} \\ & r \leq x \leq s, & & \end{array}$$

where j_i is the index from 1 to $|\mathcal{R}|$, corresponding to the order of index $i \in \mathcal{R}$.

We write the constraint function g and h of the NLP (1) as

$$g(x) = \begin{cases} a_i - c_i(x) \text{ if } i \in \mathcal{L} \\ c_i(x) - b_i \text{ if } i \in \mathcal{U} \end{cases}$$

and

$$h(x) = \begin{cases} c_i(x) - a_i \text{ if } i \in \mathcal{E} \\ c_i(x) - y_{j_i} \text{ if } i \in \mathcal{R}. \end{cases}$$

The new Lagrangian function for this model is

$$\begin{aligned} rlL(x, \lambda, \nu, y) = f(x) - \lambda_{\mathcal{L}}^T(a_{\mathcal{L}} - c_{\mathcal{L}}(x)) - \lambda_{\mathcal{U}}^T(c_{\mathcal{U}}(x) - b_{\mathcal{U}}) \\ -\nu_{\mathcal{E}}^T(c_{\mathcal{E}}(x) - a_{\mathcal{E}}) - \nu_{\mathcal{R}}^T(c_{\mathcal{R}}(x) - y). \end{aligned}$$

Defining $\lambda = (\lambda_{\mathcal{L}}, \lambda_{\mathcal{U}})$ and $\nu = (\nu_{\mathcal{E}}, \nu_{\mathcal{R}})$, the corresponding MCP model is to find $z = (x, \lambda, \nu, y) \in \mathbf{R}^q$ (where $q = n + m + |\mathcal{R}|$) that is complementary to a nonlinear vector function F from $\mathbf{R}^q \to \mathbf{R}^q$ defined as

$$F(z) := \begin{bmatrix} \nabla_x L(z) \\ a_{\mathcal{L}} - c_{\mathcal{L}}(x) \\ c_{\mathcal{U}}(x) - b_{\mathcal{U}} \\ c_{\mathcal{E}}(x) - a_{\mathcal{E}} \\ c_{\mathcal{R}}(x) - y \\ \nu_{\mathcal{R}} \end{bmatrix},$$

where $\nabla_x L(z) = \nabla_x f(x) - \lambda^T \nabla_x g(x) - \nu^T \nabla_x h(x)$, and

$$\begin{bmatrix} r \\ -\infty \\ -\infty \\ a_{\mathcal{R}} \end{bmatrix} \leq z = \begin{bmatrix} x \\ \lambda \\ \nu \\ y \end{bmatrix} \leq \begin{bmatrix} s \\ 0 \\ \infty \\ b_{\mathcal{R}} \end{bmatrix}.$$

4.2 Solver links in AMPL

AMPL executes the NLP solver as a separate program and communicates with it using the file system. Files with extension `.nl` contain a description of the model whereas files with extension `.sol` contain a termination message and the final solution written by the solver. The AMPL system uses information from these files to allocate space, generate the ASL structure and set global variable values. These values are used to identify the problem dimension, the value of objective function at the current point, the gradient evaluation, the constraint evaluation and its derivatives in sparse format.

Useful global variables are

`n_var` the total number of variables,
`n_obj` the total number of objective functions,
`n_con` the total number of constraints,
`nzc` the number of nonzeros in the Jacobian matrix and
`nzo` the number of nonzeros of the objective gradient.

The ASL structure is made up of two main components, `Edagpars` and `Edaginfo`. The `Edagpars` contains information to evaluate the objective function, constraint functions and their first and second order derivatives. The `Edaginfo` contains the upper and lower bounds, the initial point, the compressed column structure of the Jacobian matrix of the constraint functions, the pointer structure of the first order derivatives of the objective function and constraints, and information about the NLP problem. For a complete listing of all global variables and the ASL structure, the reader should consult the AMPL manual [15].

A detailed description of our implementation, called `pathnlp`, now follows. After the `solve` command is invoked in AMPL, the AMPL system generates associated NLP problem files and communicates to the `pathnlp` solver. This solver, written in the C language, automatically constructs the primal-dual formulation of the original NLP problem. It calls the PATH solver with additional options if necessary. The PATH solver runs and returns the status of the solution point via the `Path_Solved` variable and the final solution z using the `Path_FinalZ(p)` routine. The link returns these results to the AMPL system by calling `write_sol`. AMPL reports the solution back to the user who further analyzes and manipulates the model.

We now give details of how F and $\nabla_z F$ are evaluated in the link.

- Our program allocates the ASL structure by calling `ASL_alloc` with parameter `ASL_read_pfgh` which requests the AMPL to generate all first order and second order derivatives of the objective function and constraints. In addition, the flag, `want_xpi0 = 1` is set to 1 to request the initial point. The flag, `want_deriv = 1` is set to 1 to request Jacobian evaluations and Hessian evaluations.
- Our program initializes all NLP variables by calling `getstub`. It calls `jacdim` to obtain information about the Jacobian and Hessian of the objective function and constraints.
- Our program defines the MCP variable z as (x, λ, ν, y) and sets up the lower bound as $(r, -\infty, -\infty, a_{\mathcal{R}})$ and the upper bound as $(s, 0, \infty, b_{\mathcal{R}})$.
- The function evaluation of the MCP model is defined as

$$F(z) := \begin{bmatrix} \nabla_x L(z) \\ a_{\mathcal{L}} - c_{\mathcal{L}}(x) \\ c_{\mathcal{U}}(x) - b_{\mathcal{U}} \\ c_{\mathcal{E}}(x) - a_{\mathcal{E}} \\ c_{\mathcal{R}}(x) - y \\ \nu_{\mathcal{R}} \end{bmatrix}.$$

  The value of this function at the current point is kept in the vector F. To compute $\nabla_x L(z) = \nabla_x f(x) - \lambda^T \nabla_x g(x) - \nu^T \nabla_x h(x)$, the program first evaluates $\nabla_x f(x)$ at the current point by calling `objgrd`. It retrieves the sparse Jacobian matrix of c by calling `jacval` and uses `Cgrad` as the sparse matrix structures. This produces values of $c(x)$. Then it multiplies the sparse Jacobian matrix with the corresponding Lagrange multipliers and subtracts these from $\nabla_x f(x)$. The rest of the vector is computed by calling `conval` and using the appropriate multipliers of 1, -1 or 0 to generate the vector F. Then it copies the values of $\nu_{\mathcal{R}}$ for the last $|\mathcal{R}|$ elements.
- The Jacobian evaluation of the MCP (3) is given as

$$\begin{bmatrix} \nabla^2_{xx} L(z) & +\nabla_x c_{\mathcal{L}}(x) & -\nabla_x c_{\mathcal{U}}(x) & -\nabla_x c_{\mathcal{E}}(x) & -\nabla_x c_{\mathcal{R}}(x) & 0 \\ -\nabla_x c_{\mathcal{L}}(x) & 0 & 0 & 0 & 0 & 0 \\ +\nabla_x c_{\mathcal{U}}(x) & 0 & 0 & 0 & 0 & 0 \\ +\nabla_x c_{\mathcal{E}}(x) & 0 & 0 & 0 & 0 & 0 \\ +\nabla_x c_{\mathcal{R}}(x) & 0 & 0 & 0 & 0 & -I \\ 0 & 0 & 0 & I & 0 & 0 \end{bmatrix}.$$

  This computation uses the Hessian of the Lagrangian evaluation implemented in AMPL using the following form

$$\nabla^2_{xx} L(x) = \nabla^2_{xx} \left[\sum_{i=0}^{n\_obj-1} OW[i] f_i(x) + \sigma \sum_{i=0}^{n\_con-1} Y[i] c_i(x) \right],$$

  where f_i is the objective function, c_i is the constraint function, σ is a scaling factor commonly set to +1 or -1, $OW[i]$ is a scaling factor for

objective function f_i, and $Y[i]$ is Lagrange multiplier for each c_i and equals to zero when c_i is a free constraint.
To call this routine, our program sets up the scale multiplier to be 1, $OW[0] = 1$, and the scale multiplier for the sum of constraints to be negative one, $\sigma = -1$. It copies the appropriate Lagrange multipliers to Y and calls the function `sphes`. The result returns in the structure variable named `sputinfo` which is already in the compressed column vector format used by PATH. The matrix is stored as the top left corner of the MCP Jacobian matrix. The rest of the matrix is constructed using `jacval` and put it in an appropriate place in the MCP Jacobian matrix. Note that our program uses FORTRAN indices, which is a requirement for the PATH solver.

5 Results using the PATHNLP solver

We assume that a user has created a nonlinear problem using the AMPL syntax and solves it by issuing the following commands:

```
option solver pathnlp;
solve;
```

A user can guide the PATH solver using an option file, `path.opt` identified by

```
options pathnlp_options "optfile=path.opt";
```

Alternatively, the user can specify the options directly using the following syntax

```
options pathnlp_options "option_name=option_value";
```

Note that `option_name` must be a valid option of the PATH solver (see [10]). For example, to see the warning messages and current option settings of the PATH solver, a user can specify the following:

```
options pathnlp_options "output_warn=yes output_options=yes";
```

To increase the number of iterations, a user can specify

```
options pathnlp_options
"major_iteration_limit=1000 minor_iteration_limit=10000";
```

To decrease the convergence tolerance from 1×10^{-6} to 1×10^{-8}, a user can specify

```
options pathnlp_options "convergence_tolerance=1E-8";
```

Consult [10,11] for details on these and other options.

5.1 The Hock/Schittkowski test suite

We tested `pathnlp` with and without the new merit function using the Hock/Schittkowski [20] test suite. This test used 113 NLP problems, since two of the suite are incompletely specified. All problems are retrieved from the AMPL web site, `http://www.ampl.com/ampl`. The Hock/Schittkowski test suite was implemented in AMPL by Professor Robert Vanderbei.

From the 113 NLP problems, 59 problems are unconstrained nonlinear program, 48 problems have only equality constraints, while 3 problems contain range constraints and 3 problems have both equality and range constraints. We compare our results with four different NLP solvers available in AMPL, LANCELOT [4], MINOS [27], NPSOL [17] and SNOPT [16]. All solvers run using their default options. The PATH solver with the new merit function uses the weight $\gamma = 0.30$.

Table 1 shows details of these test runs on the Hock/Schittkowski test suite.

Table 1. Number of final solutions reported from each solver

Solver	`Fail`	`Infea`	`No prog`	`Iter`	`Local`	`Optimal`	`KKT`
LANCELOT	1	2	9	8	2	91	93
MINOS	0	1	0	7	11	94	105
NPSOL	7	0	2	0	8	96	104
PATH	0	0	10	0	21	82	103
PATH (merit)	0	0	5	0	20	88	108
SNOPT	0	0	2	12	4	95	99
Total	8	3	28	27	66	546	

Here `Fail` identifies the number of errors that occur because of an unexpected break from the solver, `Infea` identifies the number of solutions that are termed by the solver to be infeasible, `No prog` identifies the number of solutions that cannot be improved upon the current point by the solver, `Iter` identifies the number of solutions that the solver reached its default iteration limits, `Local` indicates the number of solutions that the solver found solutions that are different from reported global solutions, `Optimal` identifies the number of optimal solutions that are the same as reported optimal solutions, and `KKT` identifies the sum of `Local` and `Optimal`, which are KKT solutions.

The PATHNLP solver with the new merit function is very effective for solving this problem suite, solving 108 out of 113 problems. It is certainly comparable to the other NLP solvers listed here. Furthermore, the new merit function improves the robustness of the PATH code over the default version.

The test suite provides an indication of the global solution for each of the problems. Comparing these values to those found by our algorithms, the columns labeled `Local` and `Optimal` can be generated. As one can see from

the local solution column, the PATHNLP solver is more likely to find first order points that are not globally optimal for this given test problems. A more complete breakdown of the failures is given in Table 2.

Table 2. Number of nonoptimal solutions reported from each solver.

Solver	Unconstrained	Equalities	Ranges	Both	Total
LANCELOT	19	3	0	0	22
MINOS	9	10	0	0	19
NPSOL	11	5	0	0	16
PATH	23	8	0	0	31
PATH (merit)	16	6	3	0	25
SNOPT	15	3	0	0	18
Total	59	48	3	3	

It is clear that for finding globally optimal solutions, the NPSOL solver is the most effective solver, failing only 16 times.

Table 3 reports the total timing of nonoptimal and optimal solutions from each solver in seconds. Results were tested on the Sparc machine with 64 MB RAM running SunOS version 5.6.

Table 3. Total timing of nonoptimal and optimal solutions from each solver in seconds.

Solver	Nonoptimal	Optimal	Total
LANCELOT	127.52	123.24	250.76
MINOS	352.47	39.66	392.13
NPSOL	130.91	60.10	191.01
PATH	107.15	100.30	207.45
PATH (merit)	63.43	78.95	142.38
SNOPT	9.23	34.83	44.06
Total	790.71	437.08	1227.79

Table 3 shows that SNOPT uses less time to solve this problem suite. It spends only 20.95% of the total times to detect nonoptimal solutions or failures. MINOS consumes the largest times to find nonoptimal solutions but comparable to SNOPT for finding globally optimal solutions. Our PATHNLP solver with the merit function reduces the total time by 31.36% from the default version of PATH. Clearly, these problems are too small to derive many definitive conclusions on speed.

5.2 Large nonlinear programs

We selected 4 other problems as representative large scale examples from portfolio optimization, minimal surface design, nonnegative least squares and structural optimization. All problems were retrieved from the AMPL web site, `http://www.ampl.com/ampl`. Some information regarding size and numbers of (equality) constraints is given in Table 4.

Table 4. Problem dimension statistics.

Problem	Variables	Constraints	Optimal Value
`Markowitz`	1200	201	-0.526165
`Minimal`	1681	0	7.611023
`NonnegLS`	543	393	32.644706
`Structural`	13448	13488	1039.825620

Table 5 summarizes the result of our test runs on large problem sets. Results were tested on Sparc machine with 245 MB RAM running SunOS version 5.5.1.

Table 5. Total timing from each solver in seconds.

Solver	`Markowitz`	`Minimal`	`Nonnegative`	`Structural`
LANCELOT	503	106	3	*mem*
MINOS	*sup*	*sup*	*sup*	*inf*
NPSOL	538	657	191	*mem*
PATH	84	333	2	*res*
PATH (merit)	123	221	4	18,375
SNOPT	*itr*	*sup*	*sup*	*ini*

Here a keyword in the table identifies that the solver has difficulty solving this problem, where *mem* identifies that the solver could not allocate enough spaces, *sup* identifies that the solver reported the superbasics limit is too small, *itr* identifies that the solver reached its iteration limits, *inf* identifies that the solver reported problem is unbounded, *res* identifies that the solver exceeded the resource limits and *ini* identifies that the solver found the problem is infeasible due to a bad starting point. Optimal solution values from all successfully solved problems are the same for all solvers, and are reported in Table 4. Note that MINOS and SNOPT failed to solve each of these large problems, while PATHNLP with merit function solved all of them. This shows the ability of our code for handle large problem sets which is essential for solving the real world models.

6 Conclusion

It is clear from the results presented here that forming the KKT system and solving this as a complementarity problem is a viable approach for nonlinear programming. Further experimentation is required to ascertain whether a primal-dual formulation or the use of the second order information is the critical aspect. Moreover, by adapting the link code we have described in this paper, we can solve the NLP problem using other MCP solvers such as semismooth [5], or an interior point approach [33]. This will be subject of further research.

Currently the PATH solver uses a proximal point perturbation [2] to overcome singularity problems in the Jacobian matrix. This has the tendency to remove any negative curvature and may hinder progress on non-convex problems. The improvement in performance by using the composite merit function leads us to believe that further progress on this front can be achieved by (i) modification and tuning of the merit function and (ii) exploitation of negative curvature instead of using proximal point perturbation.

References

1. S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations*. PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, August 1995.
2. S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. *Mathematical Programming*, 76:533–562, 1997.
3. A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.
4. A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *LANCELOT: A Fortran package for Large–Scale Nonlinear Optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, 1992.
5. T. De Luca, F. Facchinei, and C. Kanzow. A semismooth equation approach to the solution of nonlinear complementarity problems. *Mathematical Programming*, 75:407–439, 1996.
6. S. P. Dirkse and M. C. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995.
7. B. C. Eaves. A short course in solving equations with PL homotopies. In R. W. Cottle and C. E. Lemke, editors, *Nonlinear Programming*, pages 73–143, Providence, RI, 1976. American Mathematical Society, SIAM–AMS Proceedings.
8. M. C. Ferris, R. Fourer, and D. M. Gay. Expressing complementarity problems and communicating them to solvers. Mathematical Programming Technical Report 98-02, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1998.
9. M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming, forthcoming*, 1998.

10. M. C. Ferris and T. S. Munson. Complementarity problems in GAMS and the PATH solver. Mathematical Programming Technical Report 98-12, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1998.
11. M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications, forthcoming*, 1998.
12. A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley & Sons, New York, 1968. SIAM Classics in Applied Mathematics 4, SIAM, Philadelphia, 1990.
13. A. Fischer. A special Newton–type optimization method. *Optimization*, 24:269–284, 1992.
14. R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press, 1993.
15. D. M. Gay. Hooking your solver to AMPL. Technical report, Bell Laboratories, Murray Hill, New Jersey, 1997. Revised 1994, 1997.
16. P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. Report NA 97-2, Department of Mathematics, University of California, San Diego, San Diego, California, 1997.
17. P. E. Gill, W. Murray, M. A. Saunders, and Margaret H. Wright. User's Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming. Technical Report SOL 86-2, Department of Operations Research, Stanford University, Stanford, California, January 1986.
18. A. Griewank and G. F. Corliss. Automatic differentiation of algorithms: Theory, implementation, and application. *SIAM*, pages 53–60, 1991.
19. A. Griewank, D. Juedes, and J. Utke. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Transactions on Mathematical Software*, 1996.
20. W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*, volume 187 of *Lecture Notes in Economics and Mathematical Systems.* Springer Verlag, Berlin, Germany, 1981.
21. N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.
22. W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.
23. L. G. Khachian. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.
24. H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, Berkeley and Los Angeles, 1951.
25. O. L. Mangasarian. *Nonlinear Programming.* McGraw–Hill, New York, 1969. SIAM Classics in Applied Mathematics 10, SIAM, Philadelphia, 1994.
26. B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.
27. B. A. Murtagh and M. A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83.20, Stanford University, Stanford, California, 1983.
28. L. B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120. Springer Verlag, Berlin, 1981.

29. D. Ralph. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research*, 19:352–389, 1994.
30. S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17:691–714, 1992.
31. S. M. Robinson. Newton's method for a class of nonsmooth functions. *Set Valued Analysis*, 2:291–305, 1994.
32. R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.
33. D. Shanno and E. Simantiraki. Interior point methods for linear and nonlinear programming. In I. S. Duff and G. A. Watson, editors, *State of the Art in Numerical Analysis*, Oxford, 1997. Oxford University Press.
34. S. J. Wright. *Primal–Dual Interior–Point Methods*. SIAM, Philadelphia, Pennsylvania, 1997.